AAECC **11**, 69–88 (2000)

AAECC
Applicable Algebra in Engineering, Communication and Computing


# On Short Multiplications and Divisions

**Thom Mulders**

Institute of Scientific Computing, ETH Zurich, Switzerland (e-mail: mulders@inf.ethz.ch)

Received: July 9, 1998; revised version: July 6, 1999

**Abstract.** Computing only the low degree terms of the product of two univariate polynomials is called a short multiplication. By decomposition into subproblems, a short multiplication can be reduced to appropriate addition of the results of a number of full multiplications. In this paper a new way of choosing the size of the subproblems is proposed. Computing the quotient of two polynomials is called a short division. The ideas used in the short multiplication algorithm are transferred to an algorithm for short divisions. Finally, several applications of short multiplications and divisions are pointed out.

**Keywords:** Polynomial arithmetic, Power series arithmetic, Fraction-free Gaussian elimination, Toeplitz matrices.

## 1 Introduction

In [10] efficient algorithms for multiprecision floating point multiplication are developed. The main idea of these algorithms is the fact that, in order to perform such a multiplication, it is not necessary to first perform the full exact multiplication. In fact the products of the least significant digits do (in general) not contribute to the final result. In [10] the multiprecision floating point multiplication is recursively decomposed into a number of subproblems. In this way the result is computed as the sum of a number of full products.

A similar situation occurs when one is computing in the residue ring $S = R[x]/(x^N)$, for some ring $R$ and positive integer $N$. When $F, G \in S$ are represented by $f, g \in R[x]$ (both of degree $<N$), one could compute the representative $h \in R[x]$ (of degree $<N$) for $FG$ by first computing the full product $fg$ of $f$ and $g$ and subsequently deleting the terms of degree $\geq N$. Since

$$h = \sum_{k=0}^{N-1} \left( \sum_{i+j=k} f_i g_j \right) x^k \ , \tag{1}$$

where $f_i$ (resp. $g_i$) denotes the coefficient of $x^i$ of $f$ (resp. $g$), the products of high degree coefficients do not contribute to $h$ and so it is not necessary to compute the full product $fg$.

The algorithm of [10] can easily be translated to the polynomial case and yields an algorithm, to compute $h$, that is more efficient than full multiplication. We will show that this algorithm is not optimal and is asymptotically not better than full multiplication. We will give an alternative way of choosing the size of the subproblems in the algorithm. This size will depend on the algorithm used for full multiplications and we will present our result in such a way that it is applicable for various algorithms used for full multiplications. We will show that by using this new way of choosing the subproblems, the algorithm performs better than when using the subdivision of [10].

Next we will apply the ideas used for short multiplication to polynomial division. This will improve the efficiency of division algorithms in a similar way as for short multiplication.

Since there are no manageable exact formulas for the cost functions that we need in this paper to do our analysis, we will make some simplifying assumptions. This implies that the results we get are only approximations, but still they suffice to lead us to some more efficient algorithms. To justify the use of the assumptions we will do some experiments and they will show that the results we get can be interpreted as a kind of average case result. We will also give some timings of experiments using an implementation of the algorithms described in this paper. They will show that our theoretical results are in accordance with practice.

Finally, we will give some applications of our algorithms and the gain that is yielded by using them instead of full multiplication and division algorithms.

It should be noted that the results in this paper do not improve any theoretical complexity bounds. They only improve the performance of certain tasks by a constant factor.

The way of choosing the size of the subproblems as proposed in this paper is also applicable to multiprecision integer multiplication. However, due to the phenomenon of carries and their propagation, when multiplying integers, the details are much more complicated in that case (see for example [10]). For that reason we will restrict ourselves to the polynomial case, since that will suffice to expose the main ideas.

*Remark.* A. Schönhage pointed out to the author that in [12] there is an exercise (page 35, exercise 17) concerning short multiplication using full Karatsuba multiplication. The algorithm proposed in this paper for this case was found independently of this exercise. The authors of the exercise had also this algorithm in mind but never published their solution.

## 2 Short Multiplications

From now on let $R$ be any ring. For polynomials $f, g \in R[x]$ and a non-negative integer $N$ we define the *short product* $f \times_N g$ of $f$ and $g$ with respect to $N$ (notation similar as in [10]) as:

$$f \times_N g = \sum_{k=0}^{N-1} \left( \sum_{i+j=k} f_i g_j \right) x^k \quad . \tag{2}$$

The computation of a short product is called a *short multiplication*. When using the classical polynomial multiplication algorithm, equation (2) shows how one can perform a short multiplication: compute the coefficient products that contribute to the short product and add them appropriately. It is clear how this algorithm compares to classical full multiplication. In this section we will also consider other polynomial multiplication algorithms.

We represent polynomials by sequences of coefficients, i.e. the sequence $f = (f_0, f_1, \ldots, f_{n-1})$ of elements in $R$ represents the polynomial $\sum_{i=0}^{n-1} f_i x^i$. We say that the *length* $\mathrm{L}(f)$ of $f = (f_0, f_1, \ldots, f_{n-1})$ is $n$. We then have $\deg(f) \leq n-1$ and when $f_{n-1} = 0$ we have $\deg(f) < n-1$.

First we will assume that both $f$ and $g$ have length at least $N$. The full product of $f$ and $g$ can be depicted by the left picture of Fig 1, in which the coefficient products contributing to $f \times_N g$ are indicated by the light gray area.

The right picture of Fig. 1 is a pictorial representation of the algorithm that we propose to compute $f \times_N g$. We first compute the full product of the "lower parts" of $f$ and $g$ of size $\beta N$ (the dark gray area) and then recursively compute two short products of polynomials of smaller length (the two light gray areas). Next we add appropriate terms to get the final result. For the computation of the full products we can use any multiplication algorithm. The number $1/2 \leq \beta < 1$ will depend on the algorithm used for the full multiplications and will be determined later. Notice that when $\beta > 1/2$ we do incorporate some useless coefficient products in the full product of the "lower parts". Still, by decomposing the problem in this way, the algorithm for short multiplication we get is more efficient than the full multiplication algorithm.

Now we will give a precise description of the algorithm. For a polynomial $f$ and non-negative integers $r$ and $s$, we denote by $f^{[r,s]}$ the polynomial

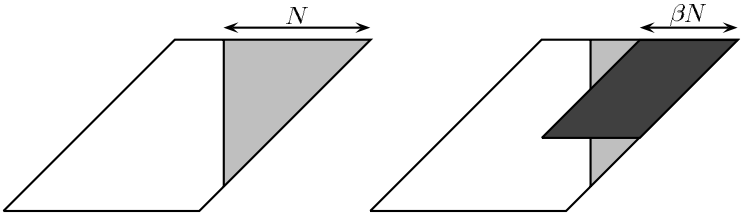


**Fig. 1.** Computing a short product ($N \leq \mathrm{L}(f)$, $N \leq \mathrm{L}(g)$)

$$f^{[r,s]} = \sum_{i=r}^{s} f_i x^{i-r} \quad .$$

Notice that when $r > s$, then $f^{[r,s]} = 0$.

**Algorithm** ShortMultiplication
**Input:** $f, g \in R[x]$, $N$ a non-negative integer.
**Output:** $f \times_N g$.
**if** $N = 0$ **then**
  **return** (0)
**fi**;
$s := \lceil \beta N \rceil$;
$h := (f^{[0,s-1]} g^{[0,s-1]})^{[0,N-1]}$; (full multiplication)
$u :=$ ShortMultiplication $(f^{[0,N-s-1]}, g^{[s,N-1]}, N-s)$;
$v :=$ ShortMultiplication $(f^{[s,N-1]}, g^{[0,N-s-1]}, N-s)$;
**return** $(h + x^s u + x^s v)$

Notice that in algorithm ShortMultiplication the computation of the short product is reduced to appropriate addition of the full products of several subpolynomials of $f$ and $g$.

**Lemma 1** *Algorithm ShortMultiplication is correct.*

*Proof.* Since only the coefficients of $1, x, \ldots, x^{N-1}$ of $f$ and $g$ are involved we may assume that $f$ and $g$ have length $N$. Then, since $N - s - 1 < s$, we can write

$$f = f^{[0,s-1]} + x^s f^{[s,N-1]} = f^{[0,N-s-1]} + x^{N-s} f^{[N-s,s-1]} + x^s f^{[s,N-1]}$$

and

$$g = g^{[0,s-1]} + x^s g^{[s,N-1]} = g^{[0,N-s-1]} + x^{N-s} g^{[N-s,s-1]} + x^s g^{[s,N-1]} \quad .$$

Writing $f^{[0,N-s-1]} g^{[s,N-1]} = u + x^{N-s}\hat{u}$ and $f^{[s,N-1]} g^{[0,N-s-1]} = v + x^{N-s}\hat{v}$, where $u$ and $v$ are as in the algorithm, we get

$$\begin{aligned} fg &= f^{[0,s-1]} g^{[0,s-1]} + x^s f^{[0,N-s-1]} g^{[s,N-1]} + x^N f^{[N-s,s-1]} g^{[s,N-1]} \\ &\quad + x^s f^{[s,N-1]} g^{[0,N-s-1]} + x^N f^{[s,N-1]} g^{[N-s,s-1]} + x^{2s} f^{[s,N-1]} g^{[s,N-1]} \\ &= f^{[0,s-1]} g^{[0,s-1]} + x^s u + x^N \hat{u} + x^s v + x^N \hat{v} + x^N f^{[N-s,s-1]} g^{[s,N-1]} \\ &\quad + x^N f^{[s,N-1]} g^{[N-s,s-1]} + x^{2s} f^{[s,N-1]} g^{[s,N-1]} \quad . \end{aligned}$$

Since $2s > N-1$ and $\deg(x^s u), \deg(x^s v) \leq N-1$ it follows that $(fg)^{[0,N-1]} = \left(f^{[0,s-1]} g^{[0,s-1]}\right)^{[0,N-1]} + x^s u + x^s v$. This proves the lemma. □

We will now compare the cost of algorithm ShortMultiplication with the cost of full multiplication. We will express the cost of an algorithm in the number of

multiplications in $R$ executed by that algorithm, assuming that they contribute the major part of the cost.

Let $\mathrm{FM}(N)$ (Full Multiplication) denote the cost of some polynomial multiplication algorithm $M$ to compute the product of two polynomials of length $N$. Furthermore, let $\mathrm{SM}_\beta(N)$ (Short Multiplication) denote the cost of algorithm ShortMultiplication to compute the short product $f \times_N g$, using $M$ for the full multiplications.

We have the following relations for the cost function $\mathrm{SM}_\beta(N)$:

$$\begin{cases} \mathrm{SM}_\beta(N) = \mathrm{FM}(\lceil \beta N \rceil) + 2\,\mathrm{SM}_\beta(N - \lceil \beta N \rceil); \\ \mathrm{SM}_\beta(0) = 0 \ . \end{cases} \tag{3}$$

By applying equation (3) recursively and substituting exact values for $\mathrm{FM}(N)$ one can compute the exact value of $\mathrm{SM}_\beta(N)$ for any given $N$. However, it seems impossible to derive in this way a manageable exact formula for $\mathrm{SM}_\beta(N)$ for the following reasons:

- For most full multiplication algorithms there is no manageable exact formula for $\mathrm{FM}(N)$.
- The size of the full multiplications into which the short multiplication is subdivided depends heavily on the actual value of $N$.

So, in order to be able to perform a comparison between $\mathrm{SM}_\beta(N)$ and $\mathrm{FM}(N)$ for general $N$, we will make the following simplifications:

- We will approximate $\mathrm{FM}(N)$ by some exact formula.
- We will extend all formulas to non-integer arguments.
- We will approximate equation (3) by

$$\begin{cases} \mathrm{SM}_\beta(N) = \mathrm{FM}(\beta N) + 2\,\mathrm{SM}_\beta((1-\beta)N); \\ \mathrm{SM}_\beta(N) = 0 \qquad \text{for } N < 1 \ . \end{cases} \tag{4}$$

Of course, using these simplifications implies that the results we will derive are only approximations. However, they are still good enough to lead us to some algorithms that in practice perform better than previous algorithms. Moreover, we will show by some experiments that our theoretical results can be interpreted as a kind of average case results.

In our further analysis we will take $\mathrm{FM}(N) = N^\alpha$ for some $1 < \alpha \leq 2$. This is supposed to approximate the cost of classical multiplication for $\alpha = 2$, Karatsuba multiplication for $\alpha = \log_2(3)$ and the generalizations of Karatsuba multiplication for $\alpha = \log_2(2r+1)/\log_2(r+1)$, $r \geq 1$ (see [8], 4.3.3 and [12], 1.3.5).

By applying equation (4) recursively we get for $1 \leq x < 1/(1-\beta)$ and $k \geq 0$:

$$\begin{aligned}\mathrm{SM}_\beta(x/(1-\beta)^k) &= \sum_{i=0}^{k} 2^i \mathrm{FM}(\beta x/(1-\beta)^{k-i}) \\ &= \frac{(\beta x)^\alpha}{(1-\beta)^{k\alpha}} \sum_{i=0}^{k} (2(1-\beta)^\alpha)^i \\ &= \frac{(\beta x)^\alpha}{(1-\beta)^{k\alpha}} \frac{1-(2(1-\beta)^\alpha)^{k+1}}{1-2(1-\beta)^\alpha} \quad . \end{aligned} \tag{5}$$

Since $2(1-\beta)^\alpha < 1$ we get for large $k$:

$$\begin{aligned}\mathrm{SM}_\beta(x/(1-\beta)^k) &\approx \frac{\beta^\alpha}{1-2(1-\beta)^\alpha} \left(\frac{x}{(1-\beta)^k}\right)^\alpha \\ &= \frac{\beta^\alpha}{1-2(1-\beta)^\alpha} \mathrm{FM}(x/(1-\beta)^k) \quad , \end{aligned}$$

and thus for large $N$ we have

$$\mathrm{SM}_\beta(N) \approx A_\alpha(\beta)\mathrm{FM}(N) \quad , \tag{6}$$

where

$$A_\alpha(\beta) = \frac{\beta^\alpha}{1-2(1-\beta)^\alpha} \quad .$$

We will now give another justification of equation (6). Since $2(1-\beta)^\alpha < 1$ we get from equation (5) for sufficiently large $l$ and $k \geq l$:

$$\begin{aligned}\mathrm{SM}_\beta(x/(1-\beta)^k) &\approx \frac{(\beta x)^\alpha}{(1-\beta)^{k\alpha}} \sum_{i=0}^{l} (2(1-\beta)^\alpha)^i \\ &= \sum_{i=0}^{l} 2^i \mathrm{FM}(\beta x/(1-\beta)^{k-i}) \quad , \end{aligned}$$

so for sufficiently large $N$ we have

$$\mathrm{SM}_\beta(N) \approx \sum_{i=0}^{l} 2^i \mathrm{FM}(\beta(1-\beta)^i N) \quad .$$

So, algorithm ShortMultiplication reduces the short multiplication to a number of full multiplications of different size and a few of those full multiplications (of largest size) will dominate the cost of the algorithm. Since for $a > 0$ the fraction $\mathrm{FM}(aN)/\mathrm{FM}(N) = a^\alpha$ does not depend on $N$, it follows that for sufficiently large $N$:

$$\frac{\mathrm{SM}_\beta(aN)}{\mathrm{SM}_\beta(N)} \approx a^\alpha \quad . \tag{7}$$

The same arguments will be used later to justify a similar relation for other cost functions. From equations (4) and (7) we now get

$$\mathrm{SM}_\beta(N) \approx \mathrm{FM}(\beta N) + 2(1-\beta)^\alpha \, \mathrm{SM}_\beta(N) \quad .$$

From this equation (6) follows.

We can now minimize $\mathrm{SM}_\beta(N)$ by minimizing $A_\alpha(\beta)$, which is minimal for $\beta_{\min} = 1 - (1/2)^{(1/(\alpha-1))}$. Figure 2 plots $\beta_{\min}$ and $A_\alpha = A_\alpha(\beta_{\min})$ for $1 < \alpha \leq 2$.

We will now have a closer look at some special values of $\alpha$.

1. Classical multiplication ($\alpha = 2$):
   We have $\beta_{\min} = 1/2$ and $A_2 = 1/2$, so compared with full multiplication we gain 50%, which is not very surprising. Notice that the computation of a short product by computing only the coefficient products needed and adding them appropriately, performs the same number of multiplications as our algorithm does, but will in practice be faster than our algorithm due to less overhead.
2. Karatsuba multiplication ($\alpha = \log_2(3) \approx 1.585$):
   We have $\beta_{\min} \approx 0.694$ and $A_{\log_2(3)} \approx 0.808$, so we gain about 19%.
   The polynomial version of the algorithm in [10] takes $\beta = 1/2$. Since $A_{\log_2(3)}(1/2) = 1$, this algorithm would gain nothing in the asymptotic case. However, for moderate length polynomials this algorithm will in fact yield some gain, but this tends to 0 very rapidly for increasing length polynomials (see also [10], Section 8).
3. Generalizations of Karatsuba multiplication ($\alpha = \log_2(2r+1)/\log_2(r+1)$):
   For increasing $r$ we have that $\alpha$ approaches 1, in which case also $A_\alpha$ approaches 1. So the gain of our algorithm compared to full multiplication decreases for increasing $r$. For example, for $r = 2$ we have $\alpha = \log_2(5)/\log_2(3) \approx 1.465$ and $A_\alpha \approx 0.888$, so a gain of only 11%. Notice that these algorithms become impractical for larger values of $r$ due to their large overhead (see also [8] and [12]).

When we take for $M$ an algorithm based on fast Fourier transforms (see [3], [7] and [11]) we have $\mathrm{FM}(N) = N\log_2(N)$ or $\mathrm{FM}(N) = N\log_2(N)\log_2(\log_2(N))$. By approximating $\mathrm{FM}(N)$ by $N^\alpha$ for $\alpha$ close to 1 and noting that then $A_\alpha \approx 1$, we see that there is no gain compared to full multiplication in this case.

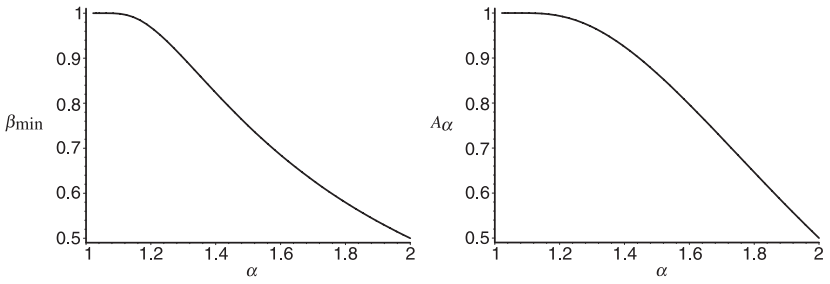


**Fig. 2.** $\beta_{\min}$ and $A_\alpha$ for $1 < \alpha \leq 2$

Now we will consider the case when $f$ and/or $g$ have length less than $N$. We can then indicate the short product of $f$ and $g$ by the light gray area in Fig. 3. In this case we simply consider both $f$ and $g$ to be of length $N$ and apply algorithm ShortMultiplication. This can be depicted by the lined area in Fig. 3. Doing this will not be as costly as when $f$ and $g$ would have length $N$, since eventually polynomials to be multiplied will be 0 (the lined but not light gray area in Fig. 3).

Instead of computing only the coefficients of the low degree terms of a product of polynomials, one can also compute only the coefficients of the high degree terms. We call this the *opposite short product*. For polynomials $f, g \in R[x]$ of degree $d_f$ resp. $d_g$ it is defined by:

$$f \times^N g = \sum_{k=N}^{d_f+d_g} \left( \sum_{i+j=k} f_i g_j \right) x^k \ . \tag{8}$$

In order to compute the opposite short product one can adjust the algorithm for short multiplication or use the formula

$$f \times^N g = \mathrm{rev}(\mathrm{rev}(f, d_f) \times_{(d_f+d_g-N+1)} \mathrm{rev}(g, d_g), d_f + d_g) \ ,$$

where the $i$th coefficient of $\mathrm{rev}(h, n)$ is the $(n-i)$th coefficient of $h$, and use the algorithm for the short product described before. Let OppShortMultiplication be an algorithm to compute an opposite short product using one of these methods. For the cost of OppShortMultiplication we then get the same results as for the cost of ShortMultiplication.

## 3 Short Divisions

In this section we will show how the ideas that we have used in the algorithm for computing short products can also be used to compute the quotient of two polynomials. In the descriptions of the algorithms that we will give, any algorithm for full multiplication can be used. In the analysis of the algorithms

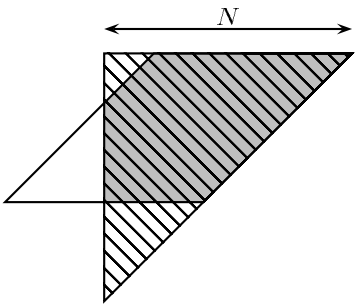


**Fig. 3.** Computing a short product ($N \geq \mathrm{L}(f)$, $N \geq \mathrm{L}(g)$)

we will take again $\mathrm{FM}(N) = N^\alpha$ for some $1 < \alpha \leq 2$. Furthermore we will assume that all divisions that have to be performed, can be performed in $R$ and we will ignore their cost (since there are far more multiplications involved).

We will call the computation of both quotient and remainder of two polynomials a *full division*. Computing only the quotient of two polynomials, will be called a *short division*.

In [6] an algorithm is described to perform a full division (in the integer case) using Karatsuba multiplication. We will now describe this algorithm in the polynomial case, using an arbitrary algorithm for full multiplication.

The algorithm can be depicted by the left picture of Fig. 4. Assume that $g$ has length $N$, $f$ has length $2N-1$ and that $g_{N-1} \neq 0$. To perform a full division of size $N$, i.e. compute $q$ and $r$ such that $f = qg + r$, where $r$ has length $N-1$, first a full division of size $N/2$ is performed (the light gray area), then a full product of size $N/2$ is computed (the diagonally lined area), subsequently another full division of size $N/2$ is performed (the dark gray area) and finally another full product of size $N/2$ is computed (the vertically lined area). We will now give a precise description of the algorithm.

**Algorithm** FullDivision
**Input:** $f, g \in R[x]$ such that $\mathrm{L}(f) = 2\,\mathrm{L}(g) - 1$, $g_{\mathrm{L}(g)-1} \neq 0$.
**Output:** $q, r \in R[x]$ such that $f = qg + r$ and $\mathrm{L}(r) = \mathrm{L}(g) - 1$.
$N := \mathrm{L}(g)$;
**if** $N = 1$ **then**
  **return** $(f_0/g_0, 0)$
**fi**;
$s := \lfloor N/2 \rfloor$;
$(q_1, r_1) :=$ FullDivision $\left(f^{[2s,2N-2]}, g^{[s,N-1]}\right)$;
$h := f^{[0,2s-1]} + x^{2s} r_1 - x^s q_1 g^{[0,s-1]}$; (full multiplication)
$(q_0, r_0) :=$ FullDivision $\left(h^{[s,2N-s-2]}, g^{[s,N-1]}\right)$;
**return** $\left(q_0 + x^s q_1, h^{[0,s-1]} - q_0 g^{[0,s-1]} + x^s r_0\right)$ (full multiplication)

**Lemma 2** *Algorithm FullDivision is correct.*

*Proof.* Since $\deg(h) \leq 2N - s - 2$ we have

$$f - (q_0 + x^s q_1)g = f^{[0,2s-1]} - q_0 g - x^s q_1 g^{[0,s-1]} + x^{2s}\left(f^{[2s,2N-2]} - q_1 g^{[s,N-1]}\right)$$

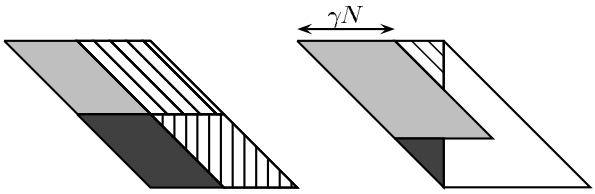


**Fig. 4.** Full and short division

$$\begin{aligned}
&= f^{[0,2s-1]} - q_0 g - x^s q_1 g^{[0,s-1]} + x^{2s} r_1 \\
&= h - q_0 g \\
&= h^{[0,s-1]} - q_0 g^{[0,s-1]} + x^s \left( h^{[s,2N-s-2]} - q_0 g^{[s,N-1]} \right) \\
&= h^{[0,s-1]} - q_0 g^{[0,s-1]} + x^s r_0 \ .
\end{aligned}$$

Since $\deg \left( h^{[0,s-1]} - q_0 g^{[0,s-1]} + x^s r_0 \right) < N - 1$, this proves the lemma. □

By $\mathrm{FD}(N)$ (Full Division) we will denote the cost (i.e. number of multiplications in $R$) of performing a full division of size $N$, using algorithm FullDivision. Making the same simplifications as in Section 2 and using similar arguments as we used to justify Eq. (7) we can argue that for $a > 0$ and sufficiently large $N$:

$$\frac{\mathrm{FD}(aN)}{\mathrm{FD}(N)} \approx a^\alpha \ .$$

We then get

$$\begin{aligned}
\mathrm{FD}(N) &= 2\,\mathrm{FD}(N/2) + 2\,\mathrm{FM}(N/2) \\
&\approx 2^{1-\alpha}\,\mathrm{FD}(N) + 2^{1-\alpha}\,\mathrm{FM}(N) \ ,
\end{aligned}$$

so $\mathrm{FD}(N) \approx C_\alpha\,\mathrm{FM}(N)$, where

$$C_\alpha = \frac{2^{1-\alpha}}{1 - 2^{1-\alpha}} \ .$$

We have $C_2 = 1$ (as expected), $C_{\log_2(3)} = 2$ (as in [6]), $C_\alpha \approx 2.63$ for $\alpha = \log_2(5)/\log_2(3)$ and $C_\alpha$ goes to $\infty$ for $\alpha$ approaching 1.

The ideas used in our algorithm to compute short products can also be used to perform a short division. We get the algorithm depicted by the right picture of Fig. 4: first perform a full division of size $\gamma N$ (the light gray area), then compute a short product of size $(1-\gamma)N$ (the diagonally lined area) and finally perform a short division of size $(1-\gamma)N$ (the dark gray area). Here $1/2 \le \gamma < 1$ will be determined later. We will now give a precise description of the algorithm.

**Algorithm** ShortDivision
**Input:** $f, g \in R[x]$ such that $\mathrm{L}(f) = 2\,\mathrm{L}(g) - 1$, $g_{\mathrm{L}(g)-1} \neq 0$.
**Output:** $q \in R[x]$ such that $\mathrm{L}(f - qg) = \mathrm{L}(g) - 1$.
$N := \mathrm{L}(g)$;
**if** $N = 1$ **then**
  **return** $(f_0/g_0)$
**fi**;
$s := \lfloor (1-\gamma)N \rfloor$;
$(q_1, r_1) :=$ FullDivision $\left( f^{[2s,2N-2]}, g^{[s,N-1]} \right)$;
$t :=$ OppShortMultiplication$(q_1, g^{[0,s-1]}, N-s-1)/x^{N-s-1}$;
$h := f^{[N-1,2s-1]} + \left( x^{2s} r_1 \right)^{[N-1,N+s-2]} - t$;
$q_0 :=$ ShortDivision $\left( x^{s-1} h, g^{[N-s,N-1]} \right)$;
**return** $(q_0 + x^s q_1)$

**Lemma 3** *Algorithm ShortDivision is correct.*

*Proof.* Writing $q_1 g^{[0,s-1]} = \hat{t} + t x^{N-s-1}$, where $t$ is as in the algorithm, we get

$$\begin{aligned} f - x^s q_1 g &= f - x^s q_1 \left( g^{[0,s-1]} + x^s g^{[s,N-1]} \right) \\ &= f^{[0,2s-1]} + x^{2s} \left( f^{[2s,2N-2]} - q_1 g^{[s,N-1]} \right) - x^s q_1 g^{[0,s-1]} \\ &= f^{[0,2s-1]} - x^s q_1 g^{[0,s-1]} + x^{2s} r_1 \\ &= f^{[0,2s-1]} - \hat{t} x^s - t x^{N-1} + x^{2s} r_1 \ . \end{aligned}$$

From this we see that $f - q_1 g x^s = \hat{h} + x^{N-1} h$, where $h$ is as in the algorithm and $\deg(\hat{h}) < N-1$. Writing $x^{s-1} h = q_0 g^{[N-s,N-1]} + r_0$, where $q_0$ is as in the algorithm, we get

$$\begin{aligned} f - (q_0 + x^s q_1) g &= \hat{h} + x^{N-1} h - q_0 g \\ &= \hat{h} + x^{N-s} \left( q_0 g^{[N-s,N-1]} + r_0 \right) - q_0 g \\ &= \hat{h} + x^{N-s} r_0 - q_0 g^{[0,N-s-1]} \ . \end{aligned}$$

From this it is clear that $\deg(f - (q_0 + x^s q_1) g) < N - 1 = \deg(g)$, which proves the lemma. □

By $\mathrm{SD}_\gamma(N)$ (Short Division) we denote the cost (i.e. number of multiplications in $R$) of performing a short division of size $N$, using algorithm ShortDivision. Again, using the same arguments as in section 2 to justify Eq. (7), we may assume that for $a > 0$ and $N$ sufficiently large we have:

$$\frac{\mathrm{SD}_\gamma(aN)}{\mathrm{SD}_\gamma(N)} \approx a^\alpha \ .$$

We then get

$$\begin{aligned} \mathrm{SD}_\gamma(N) &= \mathrm{FD}(\gamma N) + \mathrm{SM}_\beta((1-\gamma)N) + \mathrm{SD}_\gamma((1-\gamma)N) \\ &\approx (C_\alpha \gamma^\alpha + A_\alpha (1-\gamma)^\alpha)\, \mathrm{FM}(N) + (1-\gamma)^\alpha\, \mathrm{SD}_\gamma(N) \ , \end{aligned}$$

so $\mathrm{SD}_\gamma(N) \approx D_\alpha(\gamma)\, \mathrm{FM}(N)$, where

$$D_\alpha(\gamma) = \frac{C_\alpha \gamma^\alpha + A_\alpha (1-\gamma)^\alpha}{1 - (1-\gamma)^\alpha} \ .$$

Let $D_\alpha(\gamma)$ be minimal for $\gamma = \gamma_{\min}$. We are not able to give a closed form expression for $\gamma_{\min}$, in terms of $\alpha$, so we have determined $\gamma_{\min}$, $D_\alpha(\gamma_{\min})$ and $D_\alpha(1/2)$ numerically for several values of $\alpha$. They are listed in Table 1. We see that the gain by taking $\gamma$ different from $1/2$ is negligible, so we take $\gamma = 1/2$ and let $D_\alpha = D_\alpha(1/2)$. In the case of Karatsuba multiplication we then have $D_{\log_2(3)} = 1.404$ and thus $\mathrm{SD}_{1/2}(N) \approx 1.4\, \mathrm{FM}(N)$. Comparing this with $\mathrm{FD}(N) \approx 2\, \mathrm{FM}(N)$, we see that when we only need the quotient of two polynomials, the short division algorithm gains 30% compared to the full division algorithm.

**Table 1.** Optimal value for $\gamma$

| $\alpha$ | $\gamma_{\min}$ | $D_\alpha(\gamma_{\min})$ | $D_\alpha(1/2)$ |
|---|---|---|---|
| 2 | 0.500 | 0.500 | 0.500 |
| $\log_2(3)$ | 0.542 | 1.397 | 1.404 |
| $\log_2(5)/\log_2(3)$ | 0.548 | 1.988 | 1.998 |
| 1.3 | 0.547 | 3.611 | 3.611 |

## 4 Experimental Justification

Since the arguments and computations in the previous sections are very coarse we will do some experiments to justify the results that we have derived. Although the simplifications made in the previous sections imply that the ratios $\mathrm{FM}(N)/N^\alpha$, $\mathrm{FM}(aN)/\mathrm{FM}(N)$, $\mathrm{SM}_\beta(N)/\mathrm{FM}(N)$, $\mathrm{SM}_\beta(aN)/\mathrm{SM}_\beta(N)$, $\mathrm{FD}(N)/\mathrm{FM}(N)$ and $\mathrm{SD}_\gamma(N)/\mathrm{FM}(N)$ are independent of the actual value of $N$, their exact values do depend heavily on $N$. However, we will show by some experiments that the theoretical results that we have derived are a kind of average case result. In our experiments we will take Karatsuba multiplication as full multiplication algorithm, i.e. $\alpha = \log_2(3)$.

The exact value of $\mathrm{FM}(N)$ can be computed by using the relation

$$\mathrm{FM}(N) = 2\mathrm{FM}(\lceil N/2 \rceil) + \mathrm{FM}(N - (\lceil N/2 \rceil) \ ,$$

and in Fig. 5 the ratio $\mathrm{FM}(N)/N^\alpha$ and its average $\left(\sum_{i=1}^{N} \mathrm{FM}(i)/i^\alpha\right)/N$ are plotted for $N = 1, 2, \ldots, 500$. We see that only when $N$ is a power of 2 we have $\mathrm{FM}(N) = N^\alpha$. In general we only have $1 \leq \mathrm{FM}(N)/N^\alpha < 1.4$. Figure 5 has the fractal property of being self-similar. Each peak in the plot is a refinement of the previous peak spread over an interval of double length. Of course this structure reflects the fact that Karatsuba multiplication has a recursion with polynomials of half the original length. We will see a similar fractal structure in some other plots in the sequel.

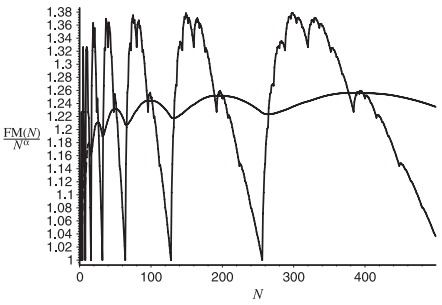


**Fig. 5.** $\mathrm{FM}(N)/N^\alpha$ and its average for $N = 1, 2, \ldots, 500$

In the previous section we could have used $cN^\alpha$ as an approximation for $\mathrm{FM}(N)$, where $c$ is some constant representing the "average" of $\mathrm{FM}(N)/N^\alpha$, but this would lead to the same results since we are only interested in ratios.

In Fig. 6 the exact values of the ratios $\mathrm{FM}(aN)/\mathrm{FM}(N)$ and $\mathrm{SM}_\beta(aN)/\mathrm{SM}_\beta(N)$ and their averages are plotted for $a = 0.8$, $\beta = 0.7$ and $N = 1, 2, \ldots, 500$. To compute the exact value of $\mathrm{SM}_\beta(N)$ one can use Eq. (3). We observe for the two ratios the same fractal structure as in Fig. 5 and one also gets the same structure for different values of $a$ and $\beta$ and for larger values of $N$. We see that the average values of $\mathrm{FM}(aN)/\mathrm{FM}(N)$ and $\mathrm{SM}_\beta(aN)/\mathrm{SM}_\beta(N)$ are close to the approximate ratio $a^\alpha \approx 0.7$ that we have derived in Section 2.

Next we plot in Fig. 7, for $\beta = 0.5$ and $\beta = 0.7$, the exact values of the ratio $\mathrm{SM}_\beta(N)/\mathrm{FM}(N)$ and their averages for $N = 1, 2, \ldots, 500$. Again we observe the same fractal structure as in previous figures. As predicted in Section 2, we see that for $\beta = 0.5$ the ratio tends to 1 and that for $\beta = 0.7$ the average value of $\mathrm{SM}_\beta(N)/\mathrm{FM}(N)$ is close to the approximate ratio $A_\alpha \approx 0.81$.

Finally, to justify our theoretical results concerning division we plot in Fig. 8 the exact values of $\mathrm{FD}(N)/\mathrm{FM}(N)$ and $\mathrm{SD}_{1/2}(N)/\mathrm{FM}(N)$ and their averages for $N = 1, 2, \ldots, 500$. The plots show the same fractal structure as in previous plots and we see that $\mathrm{FD}(N)/\mathrm{FM}(N)$ is close to 2 and that $\mathrm{SD}_{1/2}(N)/\mathrm{FM}(N)$ is close to 1.4. This is in accordance with our results in Section 3.

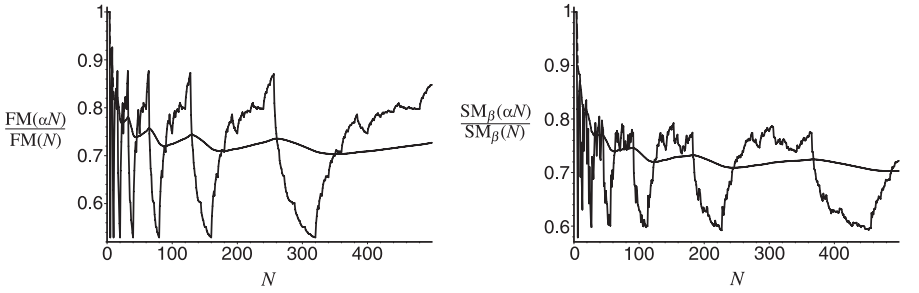


**Fig. 6.** $\mathrm{FM}(aN)/\mathrm{FM}(N)$, $\mathrm{SM}_\beta(aN)/\mathrm{SM}_\beta(N)$ and their averages for $a = 0.8$, $\beta = 0.7$ and $N = 1, 2, \ldots, 500$

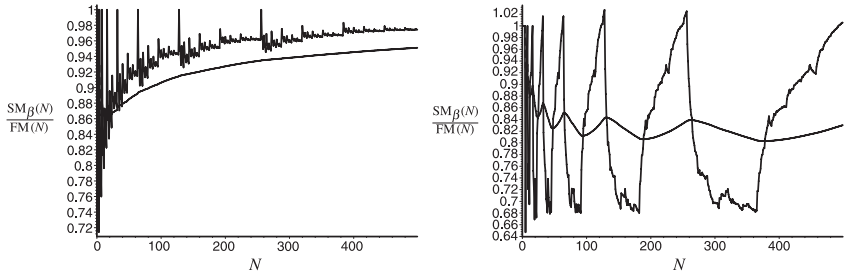


**Fig. 7.** $\mathrm{SM}_\beta(N)/\mathrm{FM}(N)$ and their averages for $N = 1, 2, \ldots, 500$ ($\beta = 0.5$ (left), $\beta = 0.7$ (right))

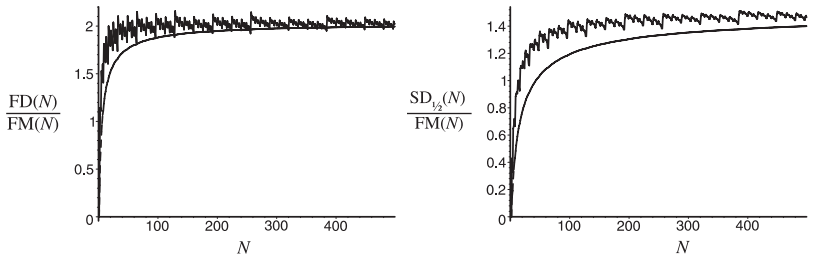


**Fig. 8.** $\mathrm{FD}(N)/\mathrm{FM}(N)$, $\mathrm{SD}_{1/2}(N)/\mathrm{FM}(N)$ and their averages for $N = 1, 2, \ldots, 500$

From the experiments described in this section we see that the theoretical results we have derived in Sections 2 and 3 can be interpreted as average case results.

## 5 Implementation

We have implemented the algorithms described in Sections 2 and 3 in Aldor as part of the $\Sigma^{\mathrm{IT}}$–library ([2]), using the built-in multiplication algorithm for Karatsuba multiplication.

In the full Karatsuba multiplication algorithm one uses classical multiplication when the length of the polynomials is less then some threshold $d$. We do the same in our implementation of short multiplication, i.e. we compute all necessary coefficient products and add them appropriately when the length of the polynomials is less then a threshold $\hat{d}$. The best choice of $\hat{d}$ depends on the coefficient domain of the polynomials and guided by some experiments we have chosen $\hat{d} = 3d$. For integer coefficient polynomials we use $d = 12$.

We have also determined the best value for $\beta$ by experiment. In this way we take into account the cost of additions and memory management that were neglected in the theoretical considerations in Section 2. Experiments show that the best choice for $\beta$ depends on $N$ and that $0.68 \leq \beta \leq 0.71$ is the best in general. So we see that the best value for $\beta$ predicted in Section 2 is also best in practice and this justifies the simplifying approximations made in Section 2.

We have done some experiments using our implementation on a 336 Mhz Sun UltraSparc with 1024 Mb main memory, running Solaris 2.6. In order to see the influence of the ratio between the cost of multiplication and addition in $R$, we have performed our experiments on polynomials with the following types of coefficients:

- 50-digit integer coefficients (ratio is 6.5).
- Integer coefficients in magnitude less than 100 (ratio is 1.2).

First we determined the ratio of the timing $t_{\mathrm{SM}}$ of short multiplication and $t_{\mathrm{FM}}$ of full multiplication for polynomials of length $N = 50, 51, \ldots, 500$. They and their averages are plotted in the left pictures of Figs. 9 and 10. Next we

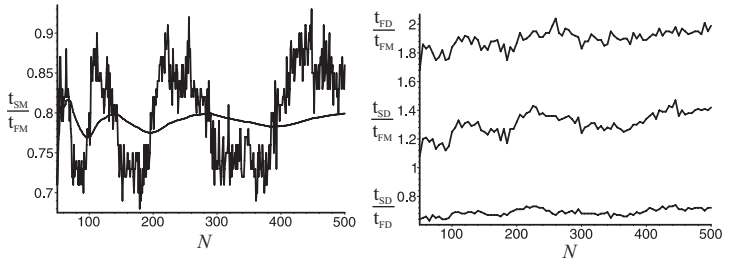


**Fig. 9.** $t_{\mathrm{SM}}/t_{\mathrm{FM}}$ (left) and $t_{\mathrm{FD}}/t_{\mathrm{FM}}$, $t_{\mathrm{SD}}/t_{\mathrm{FM}}$, $t_{\mathrm{SD}}/t_{\mathrm{FD}}$ (right) for big integer coefficients

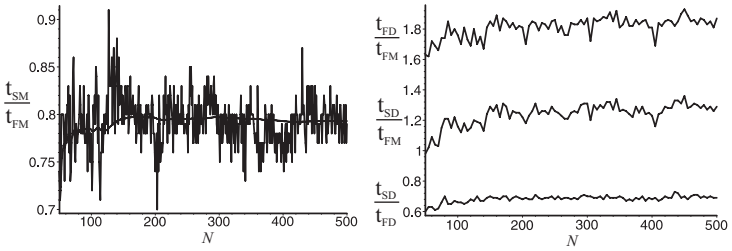


**Fig. 10.** $t_{\mathrm{SM}}/t_{\mathrm{FM}}$ (left) and $t_{\mathrm{FD}}/t_{\mathrm{FM}}$, $t_{\mathrm{SD}}/t_{\mathrm{FM}}$, $t_{\mathrm{SD}}/t_{\mathrm{FD}}$ (right) for small integer coefficients

determined the timings $t_{\mathrm{FD}}$ of full division and $t_{\mathrm{SD}}$ of short division for polynomials of length $N = 50, 55, \ldots, 500$. The ratios $t_{\mathrm{FD}}/t_{\mathrm{FM}}$, $t_{\mathrm{SD}}/t_{\mathrm{FM}}$ and $t_{\mathrm{SD}}/t_{\mathrm{FD}}$ are plotted in the right pictures of Figs. 9 and 10.

Although these plots are very irregular we see in the graphs for $t_{\mathrm{SM}}/t_{\mathrm{FM}}$ the same kind of peaks and valleys as in Fig. 7. However, they are not as extreme as in Fig. 7. For big integer coefficients the peaks and valleys are most clear. The reason for this is probably the fact that the costs of additions and overhead have a smoothing effect on the graphs and this effect is least noticeable in the big integer coefficient case because there these costs are relatively small. In both cases we see that the average of $t_{\mathrm{SM}}/t_{\mathrm{FM}}$ is close to the theoretical value $A_\alpha \approx 0.81$ derived in Section 2. For the ratios $t_{\mathrm{FD}}/t_{\mathrm{FM}}$, $t_{\mathrm{SD}}/t_{\mathrm{FM}}$ and $t_{\mathrm{SD}}/t_{\mathrm{FD}}$ we see that they are also close to the theoretical values 2, 1.4 and 0.7 that we have derived in Section 3.

We also did some timings for polynomials with unequal length. Here $f$ and $g$ have integer coefficients of magnitude less than 100, $g$ has length 50 and $f$ has length $N = 50, 51, \ldots, 500$. In Fig. 11 the ratio of $t_{\mathrm{SM}}$ and $t_{\mathrm{FM}}$ and its average are plotted. We see that when the length of $f$ is bigger than four times the length of $g$, the cost of short multiplication can exceed the cost of full multiplication. For other coefficient domains and for bigger polynomials we get the same result. Therefore we compute in practice the full product when $\mathrm{L}(f) > 4\,\mathrm{L}(g)$.

The experiments in this section show that, although we made coarse simplifying assumptions in Sections 2 and 3, the theoretical results derived there are in accordance with practice. This even holds in the case of small integer

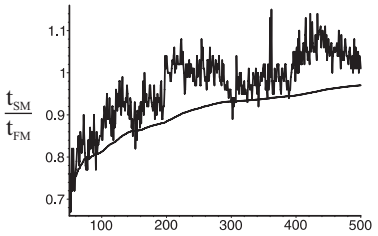


**Fig. 11.** $t_{\mathrm{SM}}/t_{\mathrm{FM}}$ and its average when $\mathrm{L}(g) = 50$ and $\mathrm{L}(f) = N$ for $N = 50, 51, \ldots, 500$

coefficients, where the ratio between the cost of coefficient multiplication and addition is small. This is because the number of additions performed by the various algorithms is proportional to the number of multiplications performed.

## 6 Applications

In this section we will give some applications of short multiplication and division. In some cases we will compute the theoretical gain obtained by replacing, where possible, full multiplication (division) by short multiplication (division). We will only consider the case $\mathrm{FM}(N) = N^{\alpha}$.

### *6.1 Exact Division*

When one knows a priori that a division of polynomials is exact, i.e. the remainder is 0, it is not necessary to compute the remainder so performing a short division suffices. However, in this case we can even further speed up the computation of the quotient.

In the algorithms of Section 3 we compute the quotient from left to right, i.e. high degree terms first. When a division of polynomials is exact one can also compute the quotient from right to left, i.e. low degree terms first (*opposite short division*), and both methods can be combined (*bidirectional division*) by computing the upper half of the quotient from left to right and the lower half from right to left (see [13] and [5], [9] for the integer case). Algorithm ShortDivision from Section 3 can be adjusted to compute the quotient from right to left. Combining both algorithms, by letting them both compute half of the quotient, we get an algorithm for exact division whose cost $\mathrm{ED}(N)$ (Exact Division) is:

$$\begin{aligned}\mathrm{ED}(N) &= 2\,\mathrm{SD}_{1/2}(N/2)\\ &= 2^{1-\alpha} D_{\alpha}\,\mathrm{FM}(N)\ .\end{aligned}$$

For classical multiplication ($\alpha = 2$) we get $\mathrm{ED}(N) = \mathrm{FM}(N)/4$, as was also shown in [13]. For Karatsuba multiplication ($\alpha = \log_2(3)$) we get $\mathrm{ED}(N) \approx 0.93\,\mathrm{FM}(N)$. Using bidirectional division in combination with full division would cost $2\,\mathrm{FD}(N/2) \approx 4/3\,\mathrm{FM}(N)$, so by using short divisions we gain 30%.

### 6.2 Power Series Division and Inversion

The task of power series division, i.e. computing for power series $p_1$ and $p_2$ a polynomial $q$ such that $q \equiv p_1/p_2 \pmod{x^N}$, can be translated to an opposite short division of size $N$. So we get for the cost PSD($N$) (Power Series Division) of power series division:

$$\mathrm{PSD}(N) = \mathrm{SD}_{1/2}(N) = D_\alpha \,\mathrm{FM}(N) \ .$$

Power series inversion, i.e. computing for a power series $p$ a polynomial $q$ such that $q \equiv 1/p \pmod{x^N}$, can be performed by power series division. However, we can do better.

Power series can be inverted by using Newtons iteration method. If $p$ is the power series to be inverted, $q_0 = 1/p(0)$ and $q_{n+1} = q_n(1 + (1 - q_n p)) \pmod{x^{2^{n+1}}}$ for $n \geq 0$, then $q_n = 1/p \pmod{x^{2^n}}$ for $n \geq 0$ (see also [4] 4.9).

Now let us look at the cost of computing $q_{n+1}$. For this we consider $p$ to be a polynomial of length $2^{n+1}$. Since $1 - q_n p = 0 \pmod{x^{2^n}}$, we only have to compute $q_n \times_{2^n} (q_n p)^{[2^n, 2^{n+1}]}$. For this, computing $q_n \times^{2^n} p^{[0,2^n]}$ and $q_n \times_{2^n} p^{[2^n, 2^{n+1}]}$ suffice to compute $(q_n p)^{[2^n, 2^{n+1}]}$. So computing (opposite) short products this computation costs only $3\,\mathrm{SM}_\beta(2^n)$.

Since the foregoing holds for all $n$ we see that, by using short multiplications, the total cost PSI($2^n$) (Power Series Inversion) of power series inversion is

$$\begin{aligned}
\mathrm{PSI}(2^n) &= \sum_{i=0}^{n-1} 3\,\mathrm{SM}_\beta(2^i) \\
&\approx 3A_\alpha \sum_{i=0}^{n-1} \mathrm{FM}(2^i) \\
&\approx \frac{3A_\alpha}{2^\alpha - 1} \mathrm{FM}(2^n) \ .
\end{aligned}$$

Since $3A_\alpha/(2^\alpha - 1) < D_\alpha$, this is a better algorithm for power series inversion than using power series division. For example, for $\alpha = \log_2(3)$ we have $3A_\alpha/(2^\alpha - 1) \approx 1.2$ while $D_\alpha \approx 1.4$.

### 6.3 Fraction Free Gaussian Elimination

When performing 1-step fraction-free Gaussian elimination on a matrix one has to perform many exact divisions of the form $(ab - cd)/q$ (see [1]). When the matrix has polynomial entries we can apply our methods in this situation. First we can use the exact division algorithm of section 6.1. When we perform an exact division $q = p_1/p_2$ of polynomials, using our method, and $q$ has length $n$, we only need to know the $n/2$ highest and $n/2$ lowest degree coefficients of $p_1$. This means that when we want to compute $(ab - cd)/q$, it is sufficient to

compute both a short and an opposite short product of $a$ and $b$, resp. $c$ and $d$ (see also [13] for the case of classical multiplication).

When the initial entries of the matrix all have degree $l$, then in general at the $k$th stage, $a$, $b$, $c$ and $d$ will have degree $kl$ and $q$ will have degree $(k-1)l$, and thus the quotient will have degree $(k+1)l$. When performing full multiplication and the bidirectional version of the full division algorithm of Section 3, the cost of each instance of computing $(ab-cd)/q$ at the $k$th stage is

$$2\,\mathrm{FM}(kl) + 2\,\mathrm{FD}((k+1)l/2) = 2\,\mathrm{FM}(kl) + 2^{1-\alpha}C_\alpha\,\mathrm{FM}((k+1)l) \ .$$

When using short products and the algorithm of Section 6.1 the cost is

$$4\,\mathrm{SM}_\beta((k+1)l/2) + \mathrm{ED}((k+1)l) = (2^{2-\alpha}A_\alpha + 2^{1-\alpha}D_\alpha)\,\mathrm{FM}((k+1)l) \ .$$

When we assume that $\mathrm{FM}((k+1)l) \approx \mathrm{FM}(kl)$ (which holds for large $k$) we go in the case of Karatsuba multiplication ($\alpha = \log_2(3)$) from $10/3\,\mathrm{FM}(kl)$ to $2\,\mathrm{FM}(kl)$, which means a gain of 40%.

Notice that in a similar way one can improve 2-step fraction-free Gaussian elimination (see [1]). In this case one has to compute expressions of the form $(ab+cd+ef)/q$ besides expressions as above.

### *6.4 Toeplitz Matrix Multiplication*

A Toeplitz matrix is a matrix of the form

$$T = \begin{bmatrix} t_0 & t_1 & \cdots & \cdots & t_{n-1} \\ t_{-1} & t_0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & t_0 & t_1 \\ t_{-(n-1)} & \cdots & \cdots & t_{-1} & t_0 \end{bmatrix} .$$

A lower (resp. upper) triangular Toeplitz matrix is a Toeplitz matrix with $t_1 = \cdots = t_{n-1} = 0$ (resp. $t_{-1} = \cdots = t_{-(n-1)} = 0$). For a lower triangular Toeplitz matrix $T_l$ and a vector $v = [v_0, \ldots, v_{n-1}]$ we have that $T_l v = [w_0, \ldots, w_{n-1}]$, where $w_i$ is the coefficient of $x^i$ in the product $(t_0 + t_{-1}x + \cdots + t_{-(n-1)}x^{n-1})(v_0 + v_1 x + \cdots + v_{n-1}x^{n-1})$. So we can compute $T_l v$ by computing a short product. In a similar way we can compute $T_u v$, where $T_u$ is an upper triangular Toeplitz matrix. Writing a Toeplitz matrix $T$ as a sum of a lower and an upper triangular Toeplitz matrix, we see that we can compute $Tv$ by computing two short products.

In the same way we see that, for a non-singular lower or upper triangular Toeplitz matrix $T$, one can compute $v$ such that $Tv = w$ by a short division. It is not clear how short multiplications and divisions can be used to compute $v$ such that $Tv = w$ for a general Toeplitz matrix.

### *6.5 Applications Involving Long Integers*

As mentioned before, the ideas used in this paper can also be applied in the case of long integer arithmetic. However, the details will become much more complicated in this case because of the phenomenon of carry propagation. In particular this will significantly complicate the algorithms that operate from left to right. To get an idea of the techniques that can be used to handle these problems the reader is referred to [9] and [10].

Also one can get an improved version of the algorithm described in [10] to perform multiprecision floating point multiplication, by using the subdivision scheme proposed in this paper. Again, the details will be more complicated due to carry propagation.

## 7 Conclusions

Translating the algorithm of [10] to the polynomial case yields an algorithm to compute short products of polynomials that is more efficient than full multiplication. This algorithm divides the short multiplication into a number of full multiplications. When one performs full multiplications by the Karatsuba multiplication algorithm, the gain of this short multiplication algorithm compared with full multiplication tends to 0 for increasing degree polynomials.

In this paper we have developed a new way of choosing the size of the full multiplications into which a short multiplication is subdivided. It is a general subdividing scheme that is applicable when one uses a full multiplication algorithm whose cost can be approximated by $\mathrm{FM}(N) = N^{\alpha}$ for some $1 < \alpha \leq 2$. The advantage of this new algorithm is that its gain compared with full multiplication is more or less constant, even for increasing degree polynomials. In the case of Karatsuba multiplication the gain is about 20%.

The ideas used to improve short multiplication are also applied to the problem of short division, i.e. computing the quotient of two polynomials. This yields an algorithm that is more efficient than computing both quotient and remainder and in the case of Karatsuba multiplication a gain of 30% is achieved.

It should be noted that the algorithms in this paper do not improve any theoretical complexity bounds, the improvements made are only constant factors. Applications of the algorithms to some frequently encountered arithmetic problems are pointed out in this paper and so they can contribute to more efficient code for many problems.

## References

1. Bareiss, E. H.: Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. Math. Comp. **22**, 565–578 (1968)

2. Bronstein, M.: $\Sigma^{IT}$ – a strongly-typed embeddable computer algebra library. Proc. DISCO'96, pp 22–33. LNCS 1128, Berlin Heidelberg New York: Springer 1996
3. Cantor, G., Kaltofen, E.: On fast multiplication of polynomials over arbitrary algebras. Acta Inf. **28**, 693–701 (1991)
4. Geddes, K. O., Czapor, S. R., Labahn, G.: Algorithms for Computer Algebra. Dordrecht: Kluwer 1992
5. Jebelean, T.: An Algorithm for Exact Division. J. Symb. Comput. **15**, 169–180 (1993)
6. Jebelean, T.: Practical Integer Division with Karatsuba Complexity. Proc. ISSAC'97, 339–341 (1997)
7. Kaminski, M.: An Algorithm for Polynomial Multiplication That Does Not Depend on the Ring Constants. J. Algorithms **9**, 137–147 (1988)
8. Knuth, D. E.: The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, 2nd edn. Reading MA: Addison-Wesley 1981
9. Krandick, W., Jebelean, T.: Bidirectional Exact Integer Division. J. Symb. Comput. **21**, 441–455 (1996)
10. Krandick, W., Johnson, J. R.: Efficient Multiprecision Floating Point Multiplication with Exact Rounding, Tech. Rep. 93-76, RISC-Linz, Johannes Kepler University, Linz, Austria 1993
11. Schönhage, A.: Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. Acta Inf. **7**, 395–398 (1977)
12. Schönhage, A., Grotefeld, A. F. W., Vetter, E.: Fast Algorithms – A multitape Turing Machine Implementation. Mannheim: BI Wissenschaftsverlag 1994
13. Schönhage, A., Vetter, E.: A New Approach to Resultant Computations and Other Algorithms with Exact Division. Proc. of the 2nd Annual European Symposium on Algorithms, pp 448–459. LNCS 855, Berlin Heidelberg New York: Springer 1994